Nuts and Bolts of Extracting Variability Models from Natural Language Requirements Documents

Eleonora Arganese[3] Alessandro Fantechi[1,2] Stefania Gnesi[2] Laura Semini[2,3]

[1] Dip. di Ing. dell'Informazione, Università di Firenze
[2] ISTI-CNR, Pisa
[3] Dipartimento di Informatica, Università di Pisa

Abstract. Natural language (NL) requirements documents are often ambiguous, and this is considered as a source of problems in the later interpretation of requirements. Ambiguity detection tools have been developed with the objective of improving the quality of requirement documents. However, defects as vagueness, optionality, weakness and multiplicity at requirements level can in some cases give an indication of possible variability, either in design and in implementation choices or configurability decisions. Variability information is actually the seed of the software engineering development practice aiming at building families of related systems, known as *software product lines*. We had therefore the idea of *hacking* an ambiguity detection tool to elicit variation points from requirement documents.
Building on the results of previous analyses conducted on large and real word requirement documents, with QuARS NL analysis tools, we provide here a classification of the forms of ambiguity that indicate variation points, and we illustrate the practical aspects of the approach by means of a simple running example.
To provide a more complete description of a line of software products, it is necessary to extrapolate, in addition to variability, also the common elements. To this end we propose here to take advantage of the capabilities of the REGICE tool to extract and cluster the glossary terms from the requirement documents.
In summary, we introduce the combined application of two different NLP tools to extract features and variability and use them to model a software product line.

Keywords: Natural Language Processing, Ambiguity in Requirements, Variability in Software Product Lines.

1 Introduction

During the first phase of software and system development, requirements are defined. This process should provide a description of the functional requirements (what the program should do) and the non-functional requirements (how the software will do it). Requirements are generally expressed in Natural Language (NL) which is intrinsically ambiguous and therefore the requirements are inherently open to different interpretations. In fact, ambiguities normally cause inconsistencies between customer expectations and the developed product, and this can lead to undesirable alterations to the final artifacts. The analysis of software requirements with respect to interpretation problems

due to the use of NL has been extensively studied in recent years [16]. A solution found within the RE community is to employ Natural Language Processing (NLP) techniques to detect them [10].

NLP is a linguistic activity focused in processing and analysing texts written in natural language, with the purpose to identify, recognize and classify the terms, and retrieve the structure, of a discourse. From the perspective of requirement analysis, NLP techniques can be used to identify those expressions which presents interpretations problems looking for lexical and syntactical constructs that may be relevant to detect ambiguities. For example we have ambiguity when a sentence contains *vague* terms having a not uniquely quantifiable meaning: i.e. terms such as clear, easy, efficient, suitable, useful etc. Another form of ambiguity is a sentence containing an *optional* part (i.e. a part that can be considered or not), that may be revealed by the presence of terms such as: in case, if possible, if appropriate, etc.. Tools have been realized to analyze requirements documents to detect ambiguities [1, 18, 21, 24, 27–29].

Software Product Line Engineering (SPLE) is a paradigm for developing a diversity of software products and software-intensive systems based on the underlying architecture of an organisation's product platform. In the context of Software Product Lines (SPLs) the introduction of variability in the software development cycle has been extensively studied [6, 8]. At all abstraction levels, a product line description is conveniently captured by a feature model, and is composed of a constant part and a variable part. The first describes those aspects that are common to all products of the family, while the latter describes those aspects, called variabilities, that are used to differentiate one product from another.

Among the fundamental activities of SPLE is the identification of the variability in different artifacts of the system, such as requirements, architecture and test cases. In particular, in the requirement engineering of SPLs, several researches have focused on exploiting NLP techniques and tools to extract information related to features and variability from requirement documents [4, 11, 19].

In a recent paper [13], we have presented the idea that often ambiguity in requirements is due to the need to postpone choices for later decisions in the implementation of the system and hence ambiguity can also be used as a way to capture variability aspects, to be solved later in the software development, envisioning an approach to achieve automated support to variability elicitation by analysing the outcomes of ambiguity detection tools. Specifically, we have used a tool developed in our Lab for analyzing NL requirements in a systematic and automatic way: QuARS (Quality Analyser for Requirements Specifications) [18]. In our process, QuARS supports an initial step for parsing NL requirements to detect potential linguistic defects that can determine interpretation problems.

Another possible application of NLP techniques is to extract information related to features from existing NL documents [3, 14, 15, 20, 22]. The idea is that of extracting candidate terms from requirements documents to build a requirement glossary and clustering them using a similarity based metric.

In this paper we propose the combined application of these two perspectives to extract both glossary terms (features) and variabilities through NLP tools and to use them to build a feature model. We focus on those cases in which ambiguity in requirements

is particularly due to the need to postpone choices for subsequent decisions in the implementation of the system. Ambiguity thus becomes a means of revealing possible points of variation in an early phase of software and system development. Then in order to enlighten commonalities, i.e. mandatory features of the SPL, we use a glossary terms extraction tool [3], accordingly to the intuition that glossary terms correspond to features, with enough precision.

To describe our proposal, we make use of an example document made of 16 requirements. The example is intentionally so small, to be analysed in details through the paper, giving preference to a qualitative analysis with respect to a quantitative one.

Structure of the paper. We start by describing some related work in Section 2. Background details are then given in Section 3, introducing feature models and an automatic requirement quality analyser (QuARS). The relation between ambiguity and variability is discussed in Section 4, exploiting the annotation produced by QuARS. Section 5 discusses some possible extension directions to improve such annotation. Section 6 describes how a glossary extraction tool can be used to identify mandatory features. In Section 7 we sketch a process for variability elicitation according to the introduced principles, and we give hints towards automatic support for such a process. Section 8 gives conclusions and future work.

2 Related Work

In [4] a systematic literature review of the state-of-the-art approaches to feature extraction from NL requirements for reuse in SPLE has been presented; this review reports on a mixture of automated and semi-automated feature clustering approaches, from data mining and information retrieval, that have been used to group common features.

In [14] the authors suggest to employ a natural language processing approach based on contrastive analysis to identify commonalities and variabilities from the textual documents describing a specific product in the railway industry (Communications-Based Train Control (CBTC) systems) in order to derive a global CBTC model, represented as a feature diagram from which specific product requirements for novel CBTC systems can be derived. The proposed method takes the brochures of different vendors as input, and identifies the linguistic expressions in the documents that can be considered as terms. In this context, a term is defined as a conceptually independent expression. The domain-specific terms that are common among all the brochures are considered as commonality candidates. On the other hand, those domain-specific terms that appear solely in a subset of the brochures are considered as variability candidates.

Another interesting proposal is in [5] where techniques capable of synthesizing feature attributes and relations among attributes have been proposed. In particular, the authors introduce an algorithmic and parametric approach for computing a legal and appropriate hierarchy of features, including feature groups, typed feature attributes, domain values and relations among these attributes starting from real-world examples.

A different approach applies product comparison matrices to identify variability patterns like optionality, multiplicity, and vagueness in tabular data [9, 25].

Finally, in [19] a different technique to analyze variability of behaviors as described in functional requirements has been presented. The approach, called semantic and ontological variability analysis (SOVA), uses ontological and semantic considerations to automatically analyze differences between initial states (pre-conditions), external events (triggers) that act on the system, and final states (post-conditions) of behaviors. The approach generates feature diagrams to model variability.

We notice however that the cited approaches are aimed at feature elicitation starting from a set of requirement documents (or other technical documentation, such as brochures), each referring to a possible variant or product, by making a comparative analysis in order to figure out common parts, assuming that the parts not in common constitute the variability.

Our idea is instead that looking at a single requirement document, the ambiguity that is present in it can be considered no more as a defect, but as a placeholder for different choices, indicating a range of different products; hence ambiguity is used to identify possible variation points. The positive role of ambiguity, when related to the need to provide a concise description of the requirements, which abstracts from irrelevant details, is discussed also in [17]: that analysis of the role of ambiguity in requirements shows how it is intimately linked to two phenomena, abstraction and absence of information, which in our view become the indicators of a possible variability.

For this purpose, a careful identification of ambiguous requirements is needed, hence we resort to a NLP based analysis, by adopting tools that have been defined for ambiguity detection in NL requirements, in order to give a preliminary classification of the ambiguity forms that can indicate variability, leaving to further analysis, and to expert judgment, a refinement of the variability identification and modelling that can be achieved with this preliminary step.

3 Background

In the context of Software Product Lines (SPLs), variability among products is made explicit by variation points, i.e., places in design artifacts where a specific decision is reduced to several features but the feature to be chosen for a particular product is left open (like optional, mandatory, or alternative features). Variety from a single product platform is achieved by identifying such variability points. Variability management is the key aspect differentiating SPLE from conventional software engineering. Modelling variability in product families has been studied extensively in the literature on SPLs, especially that concerning feature modeling [7].

3.1 Variability and Feature Models

Features and feature models have been widely used in the field of product line engineering. Product line engineering manages variability during the design process and is an important means of identifying variability needs at an early stage.

A feature, defined by the Cambridge Dictionary as “a typical quality or an important part of something”, in the context of software production is “a unit of functionality of a software system that satisfies a requirement”. It is related to a design decision, and

it provides a potential configuration option. The basic idea of feature-oriented software development, that is the base of SPL engineering is to decompose a software system in terms of the features it provides. The goal of the decomposition is to construct well-structured software that can be tailored to the needs of the user and the application scenario. Typically, from a set of features, many different software systems can be generated that share common features and differ in other features [2]

The hierarchical decomposition of a system in features and sub-features is captured by a feature model (FM). In the case of a single product, a FM simply models its decomposition structure, while in the case of SPLs, a FM also contains information on each feature being mandatory or optional, i.e. information on variability.

Feature models are visually represented by means of feature diagrams, introduced in [7] as a graphical formalism. The features are represented as the nodes of a tree and feature decomposition is rendered with the parent-child relationship. Through the paper, we use FeatureIDE, an Eclipse-based framework, to draw feature diagrams [26].

Features variability defines which features must be included in each product of the family and which ones are not. Variability is modelled with unary or n-ary predicates (graphically represented as in Figure 1):

- A **mandatory** feature F is present in a system if and only if its parent is present;
- An **optional** feature F may be present in a system only if its parent is present;
- Sibling features $F_1, F_2, \ldots$ are said to be **alternative** when one and only one is present in a system, provided their parent is present.
- Sibling features $F_1, F_2, \ldots$ are said to be **or** features if at least one is present in a system, provided their parent is present.

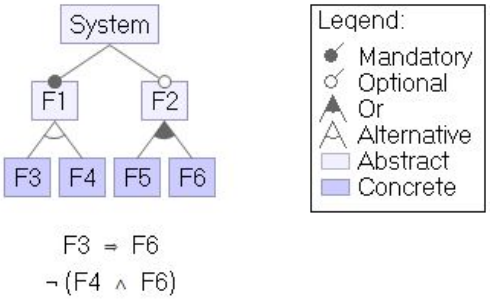


Fig. 1. Basic constructs of a feature diagram

Additional *cross-tree constraints* may be added to a feature diagram to express:

- a **requires** relation between two features, indicating that the presence of one feature implies the presence of the other ($F3 \Rightarrow F6$ in Fig. 1).
- a mutual exclusion relation between two features, called **excludes**: no system may contain the two features at the same time ($\neg\ (\ F4 \wedge F6\ )$ in Fig. 1).

Finally, a feature can be *concrete*, meaning that it will be implemented or *abstract*, in this case it is only used to group a number of features and it is not implemented.

A *Feature diagram* is hence a compact representation of the commonalities and variabilities of a family of systems, expressed as mandatory, optional features and constraints.

3.2 Quality Analysis of NL Requirements: QuARS

QuARS is a tool able to perform an analysis of Natural Language (NL) requirements in a systematic and automatic way by means of natural language processing techniques with a focus on the detection of linguistic defects. QuARS performs a linguistic analysis of a requirement document in plain text format and points out the sentences that are defective according to an expressiveness quality model, according to the process depicted in Figure 2.

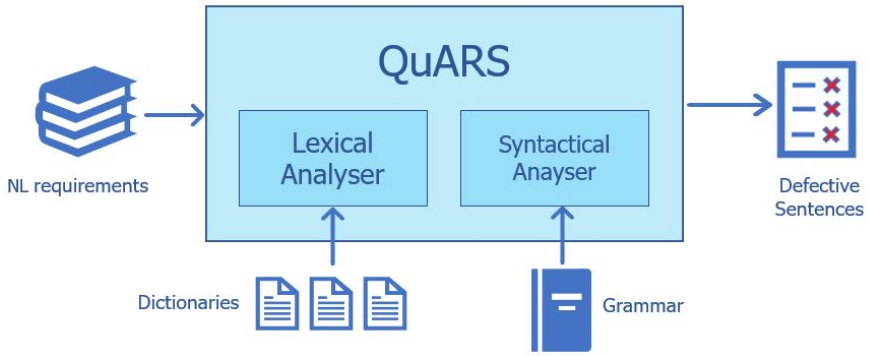


Fig. 2. QuARS Process

The defect identification process is split in two parts: (i) the ”lexical analysis” capturing *optionality, subjectivity, vagueness, weakness and lexical multiplicity* defects by identifying candidate defective words that are looked for within a corresponding set of dictionaries, that can be easily enriched with new terms if they are considered as relevant during the analysis; and (ii) the ”syntactical analysis” capturing *implicity, multiplicity and under-specification* defects.

- Optionality means that the requirement contains an optional part (i.e. a part that can or cannot be considered) and example of Optionality-revealing words are: *possibly, eventually, in case, if possible, if appropriate, if needed, ...*
- Subjectivity means that the requirement expresses personal opinions or feelings.
- Vagueness means that the requirement contains words having a no uniquely quantifiable meaning and example of Vagueness-revealing words are: *adequate, bad, clear, close, easy, far, fast, good, in front, near, recent, various, significant, slow, strong, suitable, useful, ...*

- Weakness means that the sentence contains a ”weak” verb. A verb that makes the sentence not imperative is considered weak (i.e. *can, could, may, ..*).
- Lexical multiplicity means that the requirements does not refer to a single object, but addressed a list of objects, typically using disjunctions or conjunctions (*and, or, and/or, ...*)
- Implicity means that the requirement does not specify the subject or object by means of its specific name but uses a pronoun or other indirect reference.
- Under-specification means that the requirement contains a word identifying a class of objects, without a modifier specifying an instance of this class.

In Table 1 we can see some examples of requirements that contain linguistic defects.

Table 1. Example of Requirements sentences containing defects

Indicators	**Negative Examples**
Optionality	the system shall be.., ***possibly*** without..
Subjectivity	***..in the largest extent as possible***..
Vagueness	the C code shall be ***clearly*** commented..
Weakness	the initialization checks ***may be*** reported..
Lex. Multiplicity	.. opens doors ***and*** windows..
Implicity	the ***above*** requirements shall be verified..
Under-specification	..be able to run also in case of ***attack.***

When the analysis is performed, the list of defective sentences is displayed by QuARS and a log file is created. The defective sentences can be tracked in the input requirements document and corrected, if necessary.

4 Ambiguity versus Variability

As previously said, ambiguity defects in requirements can in some cases give an indication of possible variability, either in design or in implementation choices or configurability aspects. In fact the ambiguity defects that are found in a requirement document may be due to, intentional or unintentional references made in the requirements to issues that may be solved in different ways possibly envisioning a family of different products rather than a single product. In [13] we proposed a first classification of the forms of linguistic defects that indicate variation points, and we described a possible mapping from ambiguity or under-specification defects to fragments of feature models.

We therefore use the analysis ability of QuARS to elicit the potential variability hidden in a requirement document. The process followed by QuARS for detecting potential variabilities is described below:

- A NL requirement document is given in input to QuARS to be analyzed according to the lexical and syntactical analysis provided by it looking for ambiguities.
- The detected ambiguities are analyzed in order to distinguish among false positives, real ambiguities, and variation points.

4.1 Running example

To illustrate the contribution of this paper we use a simple running example, namely a family of (simplified) e-shops, for which we consider the following requirements:

R1 The system shall enable user to enter the search text on the screen.
R2 The system shall display all the matching products based on the search.
R3 The system possibly notifies with a pop-up the user when no matching product is found on the search.
R4 The system shall allow a user to create his profile and set his credentials.
R5 The system shall authenticate user credentials to enter the profile.
R6 The system shall display the list of active and/or the list of completed orders in the customer profile.
R7 The system shall maintain customer email information as a required part of customer profile.
R8 The system shall send an order confirmation to the user through email
R9 The system shall allow an user to add and remove products in the shopping cart.
R10 The system shall display various shipping methods.
R11 The order shall be shipped to the client address or, if the "shipping to store" service is available, to an associated store.
R12 The system shall enable the user to select the shipping method.
R13 The system may display the current tracking information about the order.
R14 The system shall display the available payment methods.
R15 The system shall allow the user to select the payment method for order.
R16 After delivery, the system may enable the users to enter their reviews and ratings.

Analysing this set of requirements we notice that there are a number of defects. In the remaining part of the section we provide a classification of the forms of ambiguity that indicate variation points, to illustrate the practical aspects of our approach.

4.2 Variability due to Vagueness

Vagueness occurs whenever a requirement admits borderline cases, e.g., cases in which the truth value of the sentence cannot be decided. Running QuARS we find one vague requirement:

Example 1. (**R10**) The system shall display various shipping methods.

"Various" is a vague term, here indicating a variability about the different shipping methods that can be implemented in the e-shop system.

In general, a vague word abstracts from a set of instances, that are considered as the "various" ones, and the process of requirement refinement will make these instances explicit. In terms of features, vagueness results in the introduction of an abstract feature (Fig. 3(a)). Once instances will be made explicit, they will be represented by subfeatures, one for each instance.

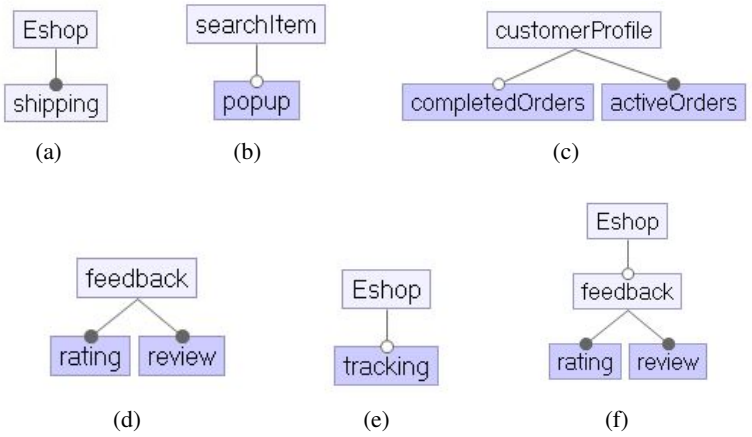


Fig. 3. (a)-(l) Feature diagram extracted fragments

4.3 Variability due to Optionality

Optionality occurs when a requirement contains an optional part, i.e. a part that can or cannot be considered. In our e-shop we find two requirements containing terms belonging to the optionality dictionary.

Example 2. (**R3**) The system possibly notifies with a pop-up the user when no matching product is found on the search.

Optionality of a requirement is naturally expressed in a feature diagram with an optional feature. In the example, it has been recognized that the pop-up notification is an optional feature, as expressed by the fragment shown in Fig. 3(b).

4.4 Variability due to Multiplicity

Another indicator of possible variability is the usage of an and/or-construct among two or more different alternatives. In particular, we have seen that the lexical multiplicity revealing words can be *and, and/or, or*, but these have different meanings, and this normally requires some expert judgment in order to define the right variability description[4].

Multiplicity due to or-constructs Again, a requirement with an or construct is not precise, since it leaves several possibilities open, and hence different products compliant to such requirement can choose different alternatives.

A disjunction may come in different flavors:

[4] In natural language, *and/or* is often used to remark that the two arguments can be present together, or only one of them, so it actually express a logical *or*. On the other hand, the usage of *or* is often implicitly intended to express a logical *exclusive or* (corresponding to alternative features in a feature diagram).

- **implicitly exclusive or:** the alternatives are mutually exclusive. In this case, the corresponding features are declared as "alternative", with a partial diagram as the one used for vagueness.
- **weak or:** all the alternatives are optional.
- **or:** at least one of the alternatives should be present, but it is irrelevant which one.
- **and/or:** as above, at least one of the alternatives should be present, but in this case there is an implicit assumption of which of the alternatives should be present in a product. This can be resolved by an analyst.

Example 3. (**R6**) The system shall display the list of active and/or the list of completed orders in the customer profile.

In this case, the interpretation of the indicated variability is that the list of active orders should be mandatory, while the list of completed orders can be considered optional (see Fig. 3(c)).

Example 4. (**R11**) The order shall be shipped to the client address or if the shipping to store service is available to an associated store.

This requirement defines the instances, "homeAddress" and "store" needed to make concrete the abstract feature "shipping" in Example 1. Their variability information is discussed in section 5.

Multiplicity due to and-constructs A conjunction, although recognized as a multiplicity, simply shows that all alternatives are mandatory: hence it is not really a variability indication, although it can be modelled with a feature diagram as well.

Example 5. (**R16**) After delivery, the system may enable the users to enter their reviews and ratings.

This is exactly the case described above, represented by the fragment in Fig. 3(d)

4.5 Variability due to Weakness

Example 6. (**R13**) The system may display the current tracking information about the order.

Example 7. (**R16**) After delivery, the system may enable the users to enter their reviews and ratings.

These two examples clearly introduce an optionality, represented in Fig. 3(e) and 3(f), respectively.

In previous works an empirical evaluation of this approach has been performed [11, 12], to validate the idea that QuARS can be hacked to extract variability: we made some experiences with six NL requirement documents of medium size, namely comprising each from 55, the smallest, to 475 requirements. The considered case studies were very

different from each other: from different domains and describing systems with very different characteristics.

The evaluation has shown that multiplicities and weak terms, such as “may” or “could” are more likely to indicate variability rather than ambiguity. A typical vagueness, “various”, normally hides a variability, while other vague terms like “useful, “significant, etc. are more likely to indicate ambiguity than variability. Finally, optional terms, such as “possibly” in most cases indicate a variability.

5 Extending QuARS to capture more variability

In [12], we also exploited the capability of QuARS to add dictionaries for new indicators: in Table II we present the new dictionaries collecting potential variability related terms and constraints identifiers (Tailored Dictionaries).

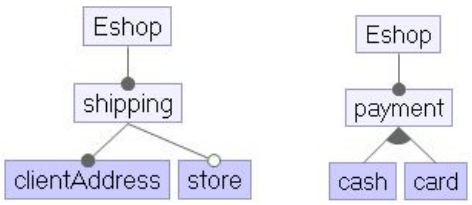
Sub-characteristic	*Indicators*
Variability	The occurrence of variability-revealing terms: if, where, whether, when, choose, choice, implement, provide, available, feature, range, select, configurable, configurate
Constraints	The occurrence of constraint-revealing terms: expect, need, request, require, use

Table 2. QuARS new Tailored Dictionary

Example 8. (**R11**) The order can be shipped to the client address or, if the “shipping to store” service is available, to an associated store.

This requirement says that the “shipping to store” service is optional, and therefore, being shipping a mandatory feature, also the clientAddress feature is mandatory, as expressed in the fragment of Fig. 4(a).

Example 9. (**R14**) The system shall display the available payment methods.



(a) (b)

Fig. 4. (a)-(b) Feature diagram extracted fragments

This is recognized to be a variability, that can be is expressed by an *or* of the different payment methods that can be adopted for the system (see Fig.4(b)). Different payment methods were not specified in the original requirements, and have been expanded in this example as payment by card or payment by cash.

Notice that in both cases above the word "available" indicates a variability, but the nature of the variability is different because in the first case it is found inside an "if" context.

During the discussion to clarify the payment methods, it was also decided that cash payment can only be done at a store, introducing a new requirement:

Example 10. (**R17**) Cash payment needs the order to be picked up in the store.

In this case the keyword "needs" appears to indicate that a *requires* cross-constraint has to be included in the model between the cash payment modality and the shipping to store feature.

Gluing together all the fragments extracted so far, we have built the feature diagram in Figure 5.

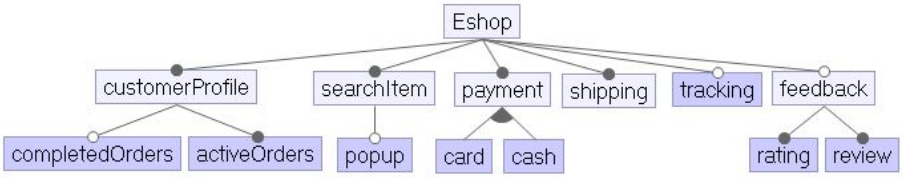


Fig. 5. Feature model built using QuARS

6 Complementing QuARS to build a complete FM

Using QuARS, we are able to build a partial feature model: indeed, focusing on ambiguities and variability, we cannot detect mandatory features that occur in the requirements in clear and assertive propositions. To this end, i.e. to complete the feature model, we look for concepts that may correspond to features, and find, by difference, the false negatives of QuARS. To this end, we use the Requirements Glossary term Identification and Clustering (REGICE) [3] tool, accordingly to the intuition that glossary terms correspond to features, with enough precision.

REGICE is a tool developed to automatically extract candidate terms of requirements glossary starting from a NL document and to organize the extracted elements into clusters of similar terms. A glossary provides a list of terms used in the requirement document characterizing an aspect relevant in software domain and it makes an explicit definition of the terms usage increasing the understandability of the text in order to mitigate ambiguity.

In [15] the Commonality Mining Tool (CMT) has been presented, which allows mining common and variant features from NL descriptions of existing products, by

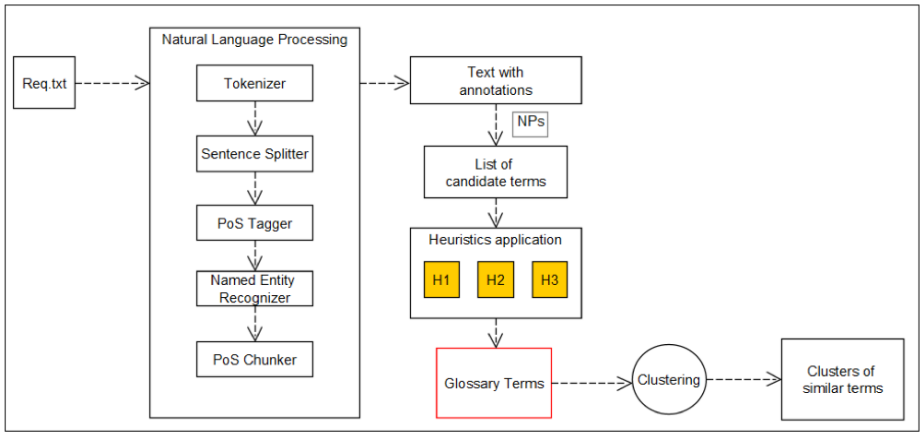


Fig. 6. REGICE Extraction Process

leveraging a NLP approach based on contrastive analysis, aimed at identifying domain-relevant terms from NL documents. Here we were more interested in analizing single documents and we have hence preferred REGICE.

When analyzing text written in natural language, several steps are required before extracting the terms of interest. These phases involve the division of the document into "tokens" or words which are first analyzed and classified at unit level and then at sentence level. The set of these phases is identified by a process that is called Text chunking [23].

The NLP pipeline used in REGICE to identify the salient terms of the glossary is shown in left side of Figure 6.

Given as input a requirement document, the text is split into words and sentences through Tokenizer and Sentence Splitter modules. Each expression is analysed using a Part of Speech (PoS) and Chunk tagger which are fundamental modules used to identify the morphological and syntactic roles of terms and phrases. PoS tagger is essential to annotate each word previously identified as a token with its part of speech function (for instance noun, verb, adjective, etc.). With PoS chunking the annotated tokens produced by PoS tagging are grouped into segments (chunks); each segment represents a specific phrase having a specific type (i.e. noun phrases-NPs, verbal phrases-VPs, etc). The document produced as output by NLP pipeline is a text containing the essential linguistic annotations. Among all detected phrases, NPs are the ones eligible as glossary terms as they express the concepts of the domain. All extracted NPs are processed and cleared using Stop word removal and lemmatization so that all elements or aspects which are considered irrelevant for the analysis (such as determiners, cardinal numbers, possessive pronouns and plural forms) are removed. The authors of [3] developed three heuristics so to reduce the number of false positive candidates provided by NLP pipeline and increase the precision of REGICE:

- H1: The combination of two NPs separated by of or a possessive s is added to the list of terms as a unique entry.

– H2: Abbreviations, acronyms and sequence of proper nouns within individual NPs are added as independent entries to the list of terms.
– H3: Common nouns appearing as single terms within individual NPs are filtered out.

The heuristics are applied to the list of candidate terms produced by NLP pipeline and the result is a list of glossary terms.

We have run REGICE on the e-shop working example, obtaining the results in Figure 7. With respect to the features singled out with QuARS (Fig. 5), REGICE was able

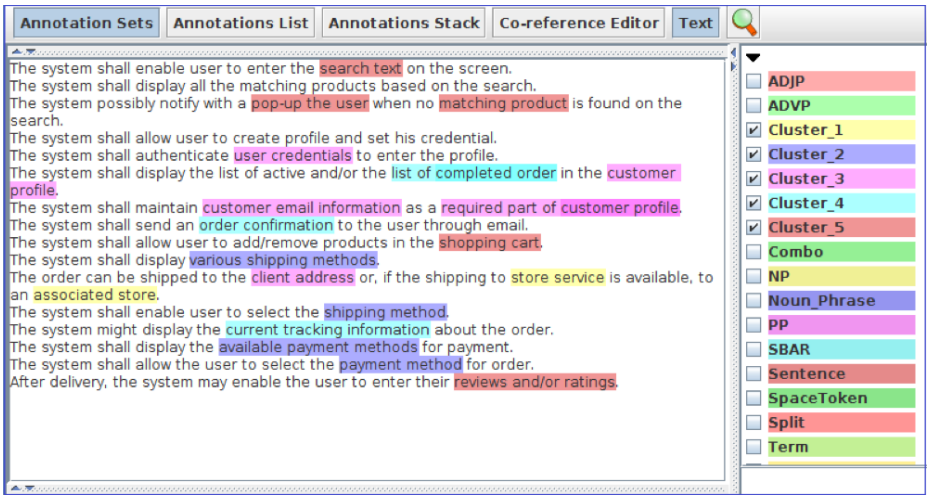


Fig. 7. REGICE Results

to detect more candidate features: matching features, user credentials, customer email information, shopping cart, which occur in requirements R2, R5, R7, R9, respectively. As expected, none of these requirements was detected as defective by QuARS, indeed they are all unambiguos requirements (with the only exception of *add/remove* in R9 which is a false negative of QuARS due to the use of the slash).

The final step is related to the cluster function where syntactic and semantic similarities are computed for each pair of terms. The purpose of this activity is to group related words in the same cluster (see Figure 8), so to underline the presence of similar terms possibly referring to the same topic. Clusters are helpful to define the relations between features in the feature model.

Using the results of he analysis with REGICE, we have built the full feature model for our case study (Fig. 9), where all the extracted fragments are combined as subfeatures of the root feature representing the e-shop.

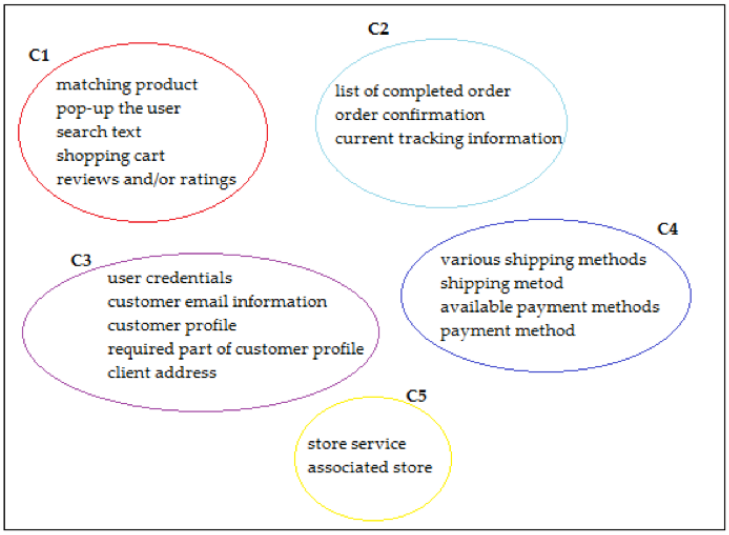


Fig. 8. REGICE Clustering

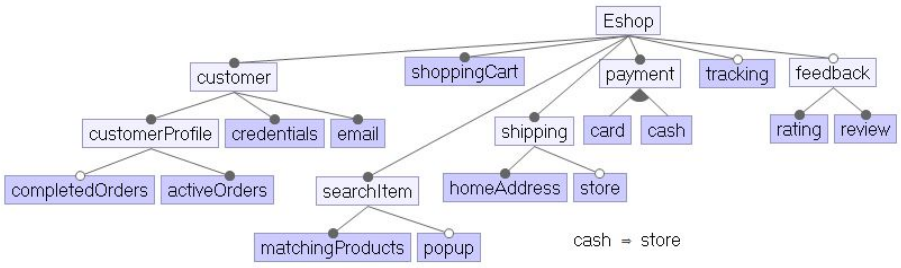


Fig. 9. Feature model completed using REGICE

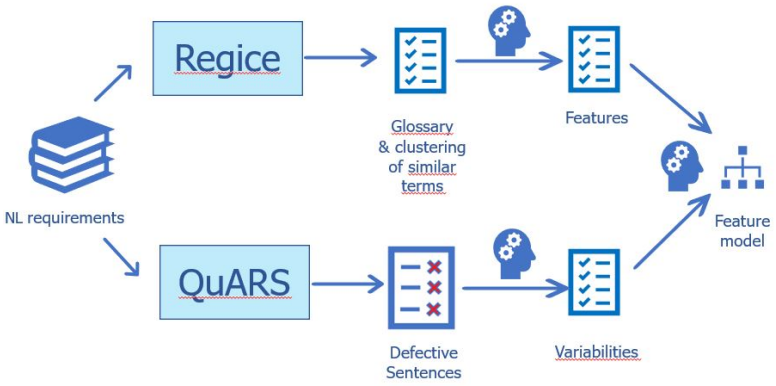


Fig. 10. Feature model elicitation process

7 The feature model elicitation process in practice

The presented approach defines a feature model elicitation process. The process, after a first requirement elicitation phase and the consequent writing of the requirements in NL, follows the steps described in Figure 10 and in the following:

1. Run an NLP tool to detect ambiguities in the requirements document. We use QuARS for this purpose.
2. Acquire expert judgment of the outcome of the tool to distinguish among false positives, actual ambiguities and variabilities.
   - A false positive is a sentence recognized as defective by the tool but considered acceptable by the judgment of an expert. The removal of false positives from the blacklist is decided by an engineer and managed by the masking functions provided by QuARS.
   - Actually ambiguous terms are those that must be substituted by more precise ones: to solve the problem clarifications are needed with the stakeholders or domain experts.
   - After having resolved the real ambiguities and masked the false positives, the intentional (or sometimes involuntary, but positive) ambiguities, due to the need to abstract from finer details, are left. These are candidates for being indicators of variability.
3. Based on the different types of defects, and based on the description given in Sections 4.2 to 4.5, and in Section 5, a feature model which captures the identified variation points is constructed, possibly requiring input from the expert.
4. The feature model is then completed adding mandatory features, using the clusters of glossary terms extracted by REGICE.

These activities could be automated by a tool that implements the entire process, this tool must integrate the existing NLP tools and support the engineer with an interactive interface in the steps in which, based on the flow indicated in Fig.10, manual

intervention is required. A tool automating the given process should in the end give the possibility to the user to edit the feature model and to add possible cross-tree constraints.

8 Conclusions

To provide a description of a line of software products, we have defined an approach to extract variability issues from a requirements document combining two Natural Language analysis tools. One of these tools, QuARS, is aimed at revealing the ambiguity defects of the NL sentences in the requirements document, since ambiguity has been recognized a means to enlighten variation points. The second tool, REGICE, is able to extrapolate and cluster glossary terms from the requirement documents, since glossary terms are good candidates for modeling features.

Other attempts have been done to use NLP to extract variability indications from a set of requirement documents (or other technical documentation, such as brochures), each referred to a possible variant, by making a comparative analysis in order to figure out common parts, assuming that the parts not in common constitute the variability. Our idea is instead that looking at a single requirement document, the ambiguity that is present in it can be used to identify possible variation points, where ambiguity is not a defect but points to different choices that can give space for a range of different products.

References

1. V. Ambriola, V. Gervasi, On the Systematic Analysis of Natural Language Requirements with CIRCE, Automated Software Engineering, Volume 13, Issue 1, pp 107-167, 2006.
2. S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013.
3. C. Arora, M. Sabetzadeh, L. Briand, F. Zimmber: Automated Extraction and Clustering of Requirements Glossary Terms, IEEE Trans. Software Eng. 43(10: 918-945, ACM 2017.
4. N. H. Bakar, Z. M. Kasirun, and N. Salleh, Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review, J. Syst. Softw., vol. 106, pp. 132-149, 2015.
5. G. Bécan, R. Behjati, A. Gotlieb, M. Acher, Synthesis of attributed feature models from product descriptions, 19th International Software Product Lines Conference, SPLC 2015: pp. 1-10.
6. Clements, P.C., Northrop, L.: Software Product Lines—Practices and Patterns. Addison-Wesley (2002)
7. Kang, K., Choen, S., Hess, J., Novak, W., Peterson, S.: Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical Report SEI-90-TR-21, Carnegie Mellon University (1990)
8. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering—Foundations, Principles, and Techniques. Springer (2005)
9. G. Bécan, N. Sannier, M. Acher, O. Barais, A. Blouin, B.Baudry, Automating the formalization of product comparison matrices, ACM/IEEE International Conference on Automated Software Engineering, ASE, 2014, pp. 433-444.

10. A. Casamayor, D. Godoy, M. Campo, "Mining Textual Requirements to Assist Architectural Software Design: A State of the Art Review", Artificial Intelligence Rev., vol. 38, no. 3, pp. 173-191, 2012.
11. A. Fantechi, A. Ferrari, S. Gnesi, and L. Semini. Hacking an ambiguity detection tool to extract variation points: an experience report. In *Proc. of the 12th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS 2018, Madrid, Feb. 2018*, pages 43–50. ACM, 2018.
12. A. Fantechi, A. Ferrari, S. Gnesi, and L. Semini. Requirement engineering of software product lines: Extracting variability using NLP. In *26th IEEE Int. Requirements Engineering Conference, RE 2018, Banff, AB, Canada, Aug. 2018*, pages 418–423. IEEE Computer Society.
13. A. Fantechi, S. Gnesi, and L. Semini. Ambiguity defects as variation points in requirements. In *Proc. of the 11th International Workshop on Variability Modelling of Software-intensive Systems*, VAMOS '17, pages 13–19, 2017. ACM.
14. A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta, Mining commonalities and variabilities from natural language documents, in Proc. 17th International Software Product Lines Conference, SPLC 2013, pp. 116-120.
15. A. Ferrari, G. O. Spagnolo, S. Gnesi, F. Dell'Orletta: CMT and FDE: tools to bridge the gap between natural language documents and feature diagrams. SPLC 2015: 402-410
16. A. Ferrari, F. Dell'Orletta, A.Esuli, V. Gervasi, S. Gnesi, Natural Language Requirements Processing: A 4D Vision, IEEE Software, to appear.
17. V. Gervasi, D. Zowghi, On the Role of Ambiguity in RE, Requirements Engineering: Foundation for Software Quality: 16th International Working Conference, (REFSQ), LNCS 6182, 2010, pp. 248-254.
18. S. Gnesi, G. Lami, G. Trentanni, An automatic tool for the analysis of natural language requirements, Comput. Syst. Sci. Eng., 20, 1, 2005.
19. N. Itzik, I. Reinhartz-Berger, Y. Wand, Variability Analysis of Requirements: Considering Behavioral Differences and Reflecting Stakeholders Perspectives IEEE Transactions on Software Engineering, 42, 7, July 2016, pp. 687-706.
20. Y. Li, S. Schulze, G.Saake: Reverse engineering variability from requirement documents based on probabilistic relevance and word embedding. SPLC 2018: 121-131
21. Mich L, Garigliano R (2000) Ambiguity measures in requirements engineering. In: Proceedings of ICS 2000 16th IFIP WCC, Beijing, China, 2125 August 2000, pp 3948
22. S. Ben Nasr, G. Bcan, M. Acher, J. Bosco Ferreira Filho, N. Sannier, B. Baudry, J.-M. Davril: Automated extraction of product comparison matrices from informal product descriptions. Journal of Systems and Software 124: 82-103
23. L. Ramshaw and M. Marcus, Text Chunking Using Transformation-based Learning, chapter in Natural Language Processing Using Very Large Corpora, Armstrong, S., Church, K., Isabelle, P., Manzi, S., Tzoukermann, E., Yarowsky, D. eds., pages 157176, Kluwer, 1999.
24. B. Rosadini, A. Ferrari, G. Gori, A. Fantechi, S. Gnesi, I. Trotta, and S. Bacherini, Using NLP to Detect Requirements Defects: an Industrial Experience in the Railway Domain, to appear on Proceedings Requirements Engineering: Foundation for Software Quality: 23rd International Working Conference, (REFSQ),2017.
25. N. Sannier, M. Acher, B. Baudry: From comparison matrix to Variability Model: The Wikipedia case study. 28th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2013, pp. 580-585.
26. T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, T. Leich: FeatureIDE: An extensible framework for feature-oriented software development. Sci. Comput. Program. 79: 70-85 (2014)

27. Wilson, W.M., Rosenberg, L.H., Hyatt, L.E.: Automated analysis of requirement specifications. In: Proceedings of the Nineteenth International Conference on Software Engineering (ICSE 1997), pp. 161171 (1997)
28. https://www.qualicen.de/en/
29. https://qracorp.com/